## Steps to Run-

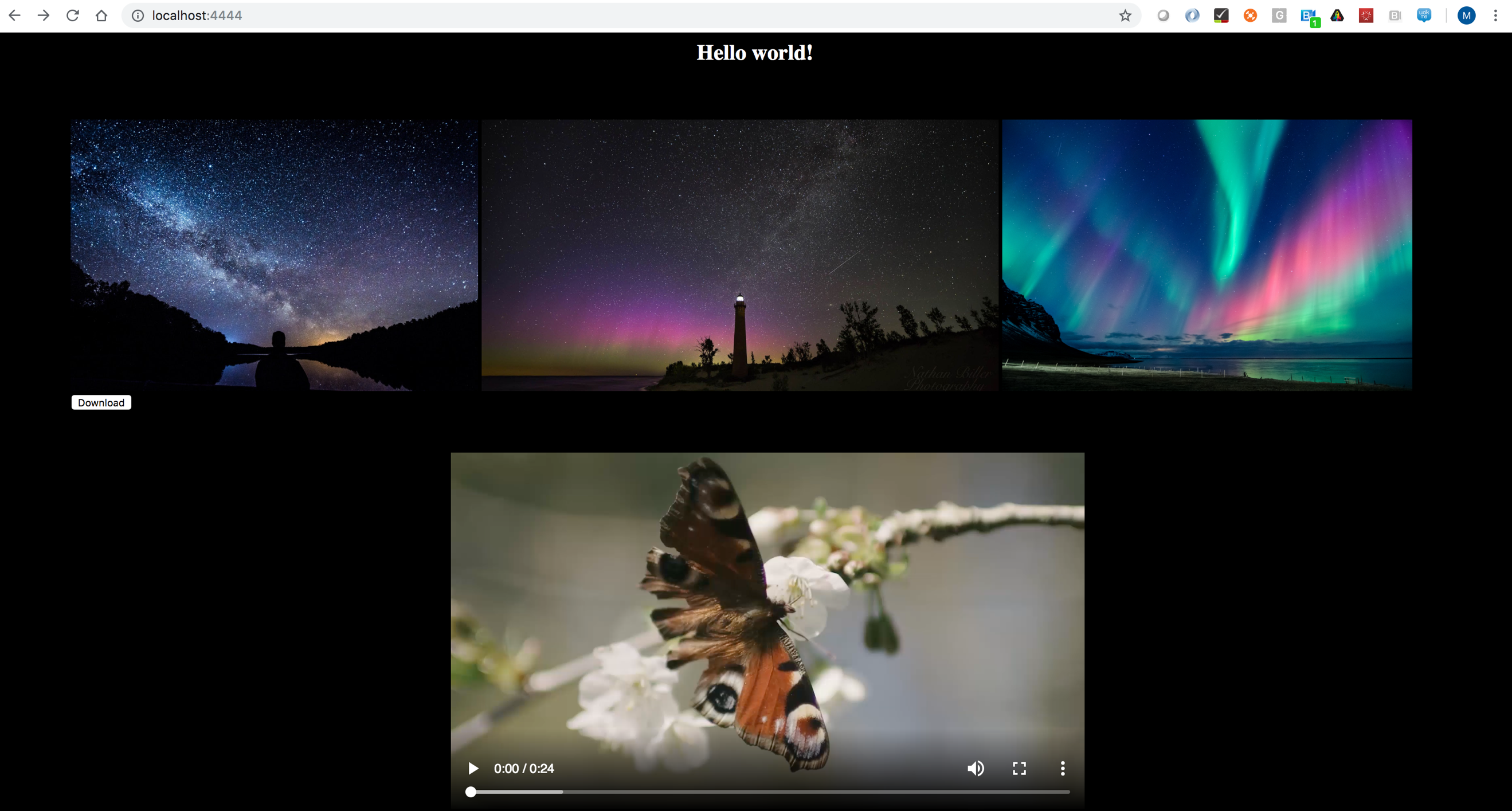
1. Clone the project from <https://github.com/manishpaul26/pizza-server>
2. Import into Eclipse/IntelliJ
3. Build the project using **mvn clean install**
4. Open the class “**PizzaServer**” and click on run
   1. You can provide the following run time arguments to the class (default in brackets)
      1. -**DpoolSize=8 –** The thread pool size
      2. -**DmaxPoolSize=20** - The Maximum thread pool size
      3. **-DqueueSize=200 -** The queue size for the Thread pool
      4. **-DsocketTimeOut=4000** – Socket time out
      5. -**DwriteToSameFile**=false – Post request concurrency behaviour
      6. **-Dverbose=false** – Print all logs- turn to true
      7. **–Dport = 4444** – Port number
5. Once you see the message “Starting Pizza server on port..”, it is ready to use.
6. Hit the url localhost:<port>
   1. Default port is localhost:4444
7. This will show the default **index.html** file.

## What all has been implemented-

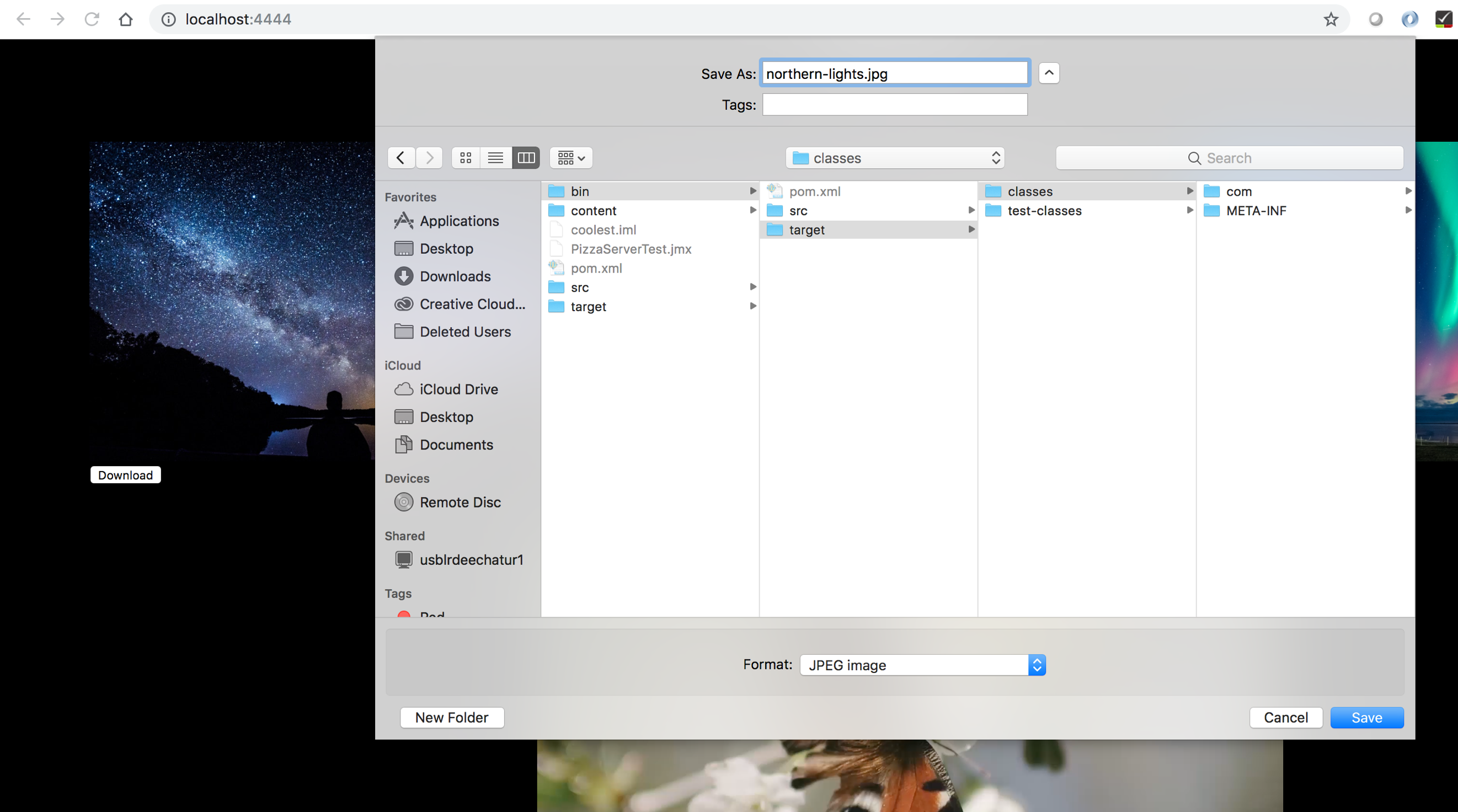
1. GET request-
   1. /index.html -> homepage
   2. /upload.html -> Page to upload images
2. POST Request-
   1. /upload.html -> Upload an image and click on “Upload Image” button. This will upload the image to the “content” folder in the repository with the name of the uploaded file.
   2. If –**DwriteToSameFile=true,** then all post requests from this page will write to the same file (**new.jpg**).
3. To solve the problem mentioned in 2(b), all writes to the same file have been synchronized by maintaining a ConcurrentHashMap of files that are currently being written. If a file has been opened for writing, no other thread can open this file until the previous thread releases the file and updates this map (**FileIO.java**).
4. Thread Pool with configurable limits
   1. Use the run time arguments to configure the thread pool
5. A JMeter script “**PizzaServerTest.jmx**” is present in the root folder. Import this into JMeter and just click on the “Run” icon to simulate multi-threaded scenario with GET and POST Requests. This script does the following-
   1. Hits the homepage – 200 OK
   2. Hits the file “content/miffy.jpg”
   3. Hits 404 paths – 404 Not Found
   4. Posts the **new.jpg** file – Enable –**DwriteToSameFile=true**
6. HTTP Implementations-
   1. 200 OK
   2. 404 Not Found
   3. Keep-Alive
   4. Content-Disposition – to download files
   5. Content Types-
      1. Images
      2. Html
      3. Icons
      4. MP4
7. **POST Requests-**
   1. A servlet framework has been created using custom Annotations.
   2. You can create your own servlet using **“@ChocoServlet**” annotation and a “**path**” parameter.
   3. During startup, all servlets with this annotation are registered and stored in a map with key as path and value as the Class itself
   4. At runtime, POST Requests check whether there is any servlet at that path. If yes, then use Reflections (**org.reflections** – dependency) to invoke the **doPost** method of that servlet.
   5. This can be seen with the “**DownloadServlet**” which is invoked on the click of the “download” button on the homepage.
8. **Performance boost-**
   1. The file miffy.jpg is cached in memory on startup to enable faster reads and avoid repeated I/O operations during the JMeter script execution. This has been done to improve the read performance when multiple threads are at work.

**Apart from basic help using Stack Overflow and other blogs, all the code has been written on my own and has not been copied.**

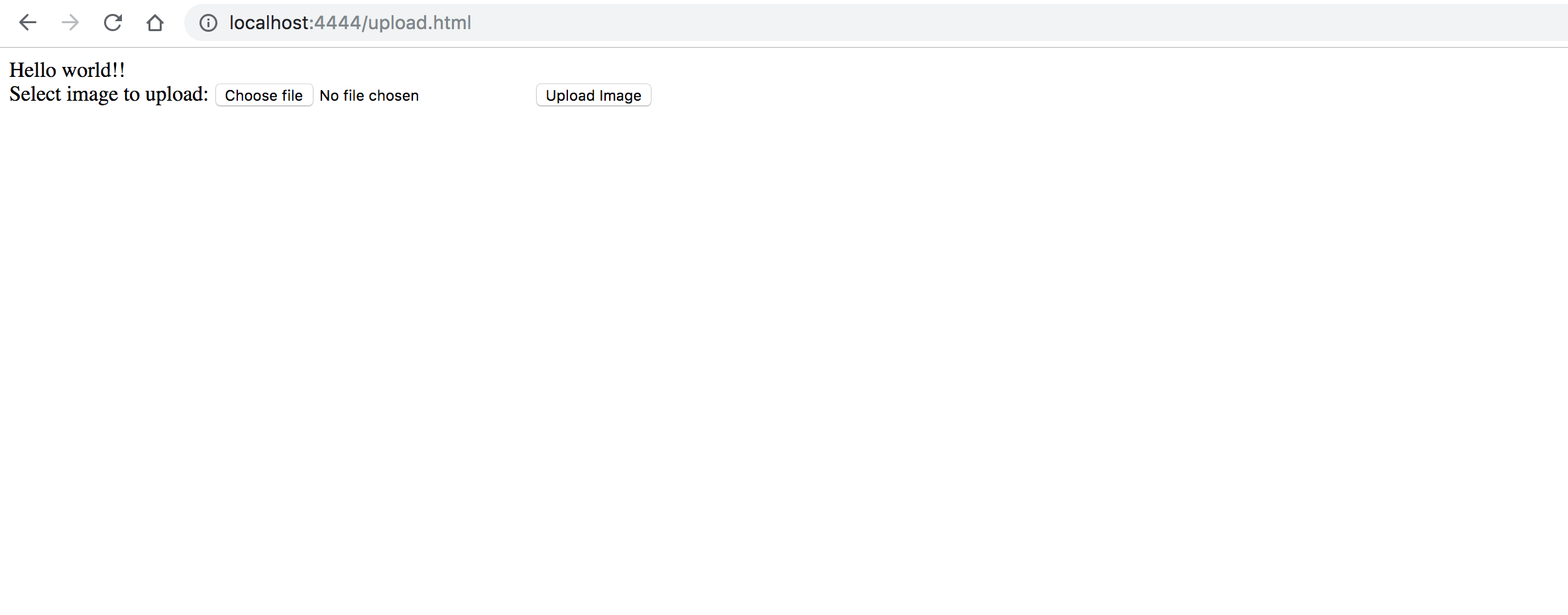
**Home Page-**



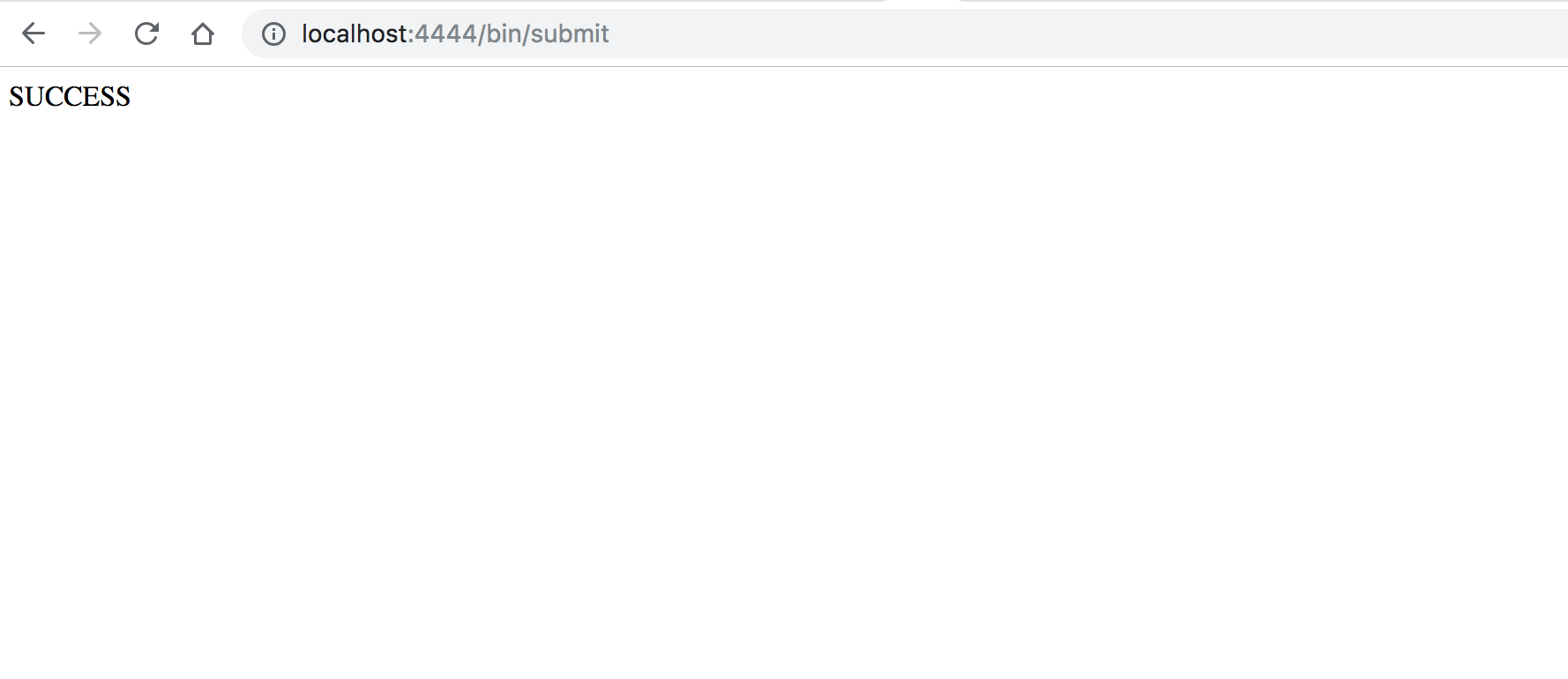
**Downloading an Image-**

****

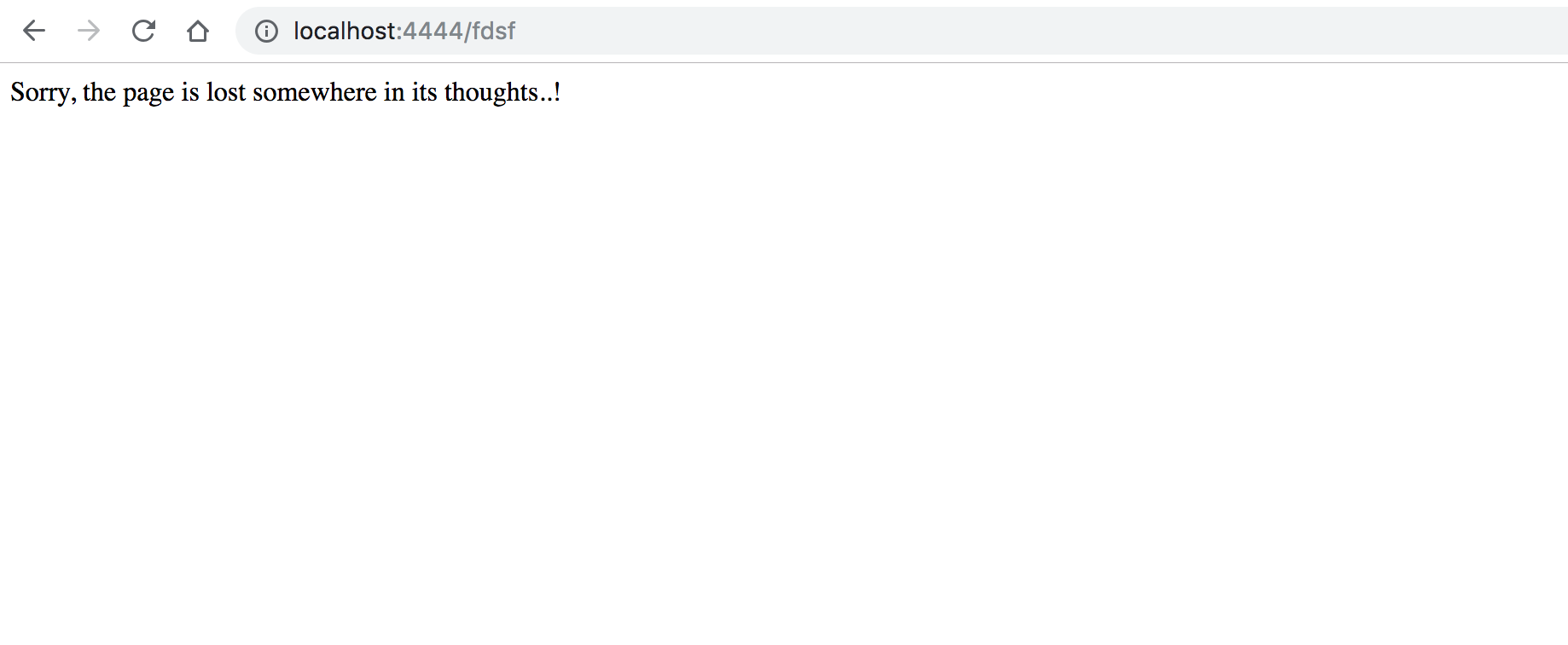
**Uploading an Image**



**Upload Successful-**



**404 Not Found-**

****